SAND2011-2450P

Intrepid Functionality for Parvis

Kara Peterson [1]
Pavel Bochev[1]
Denis Ridzal[2]

[1]Numerical Analysis & Applications
[2]Optimization & Uncertainty Quantification
Sandia National Labs

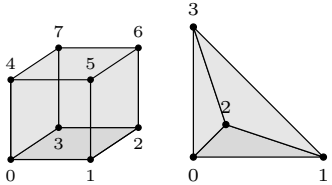
Parvis Meeting
April 11, 2011

Intrepid Functionality

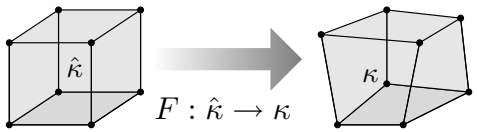
SAND2011-2450P

Cell Geometry

- Topology from Shards (line, tri, quad, hex, tet, wedge)



- Maps to and from reference cells



- Jacobians (DF, $J = \det(DF)$)
- Surface normals, line tangents, cell areas
- Tests for point inclusion

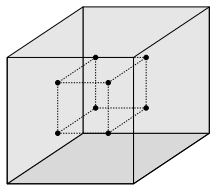
Discrete Spaces and Operators

- Nodal, edge, and face basis functions (ϕ_i)
- Discrete differential operators ($\nabla\phi_i, \nabla \times \phi_i, \nabla \cdot \phi_i, D^k\phi_i$)

$$f^h(x) = \sum_{i=1}^{N} f_i \phi_i(x)$$

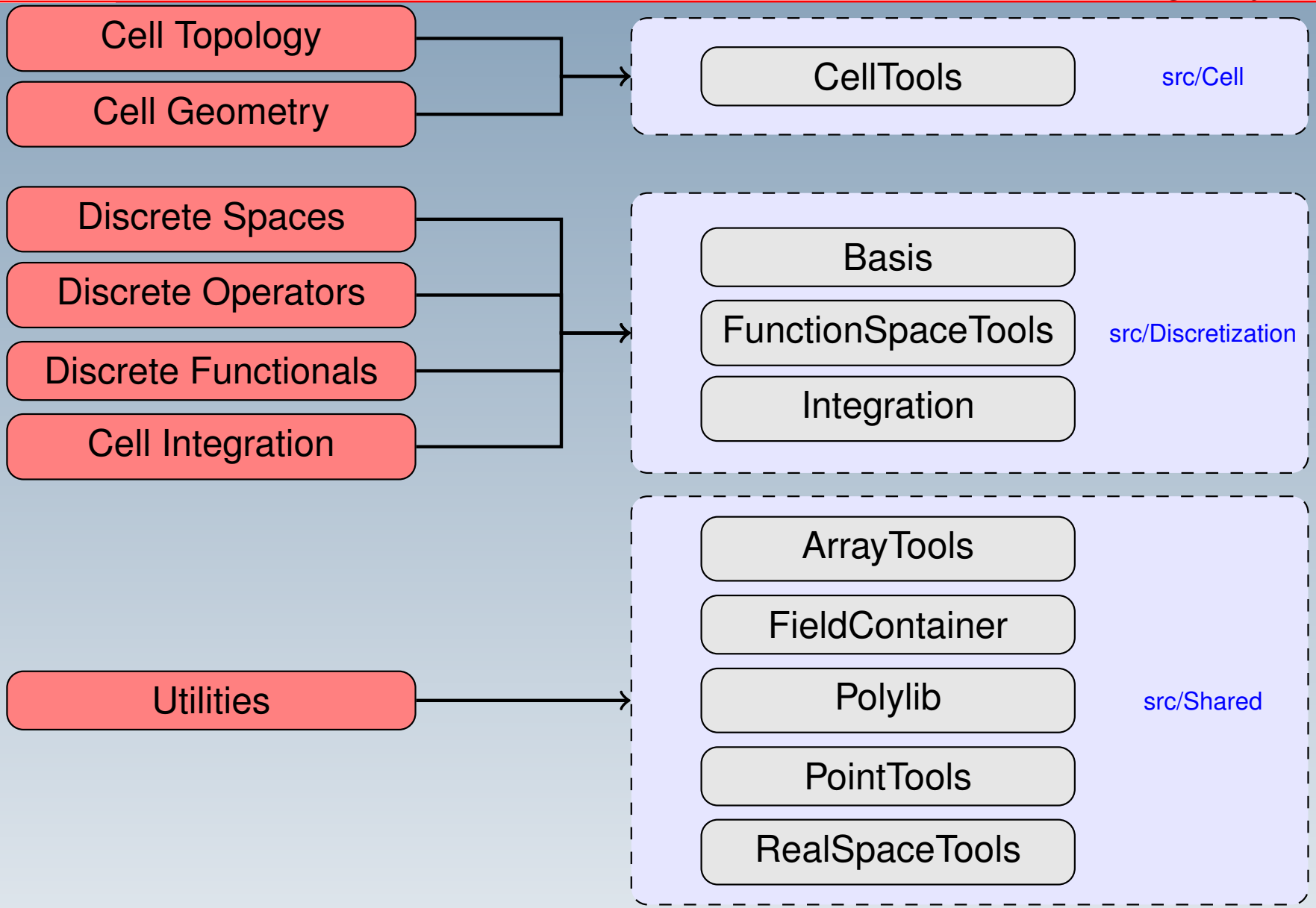
Integration

- Cubature points (x_p) and weights (w_p)



Mapping Functionality to Software

SAND2011-2450P

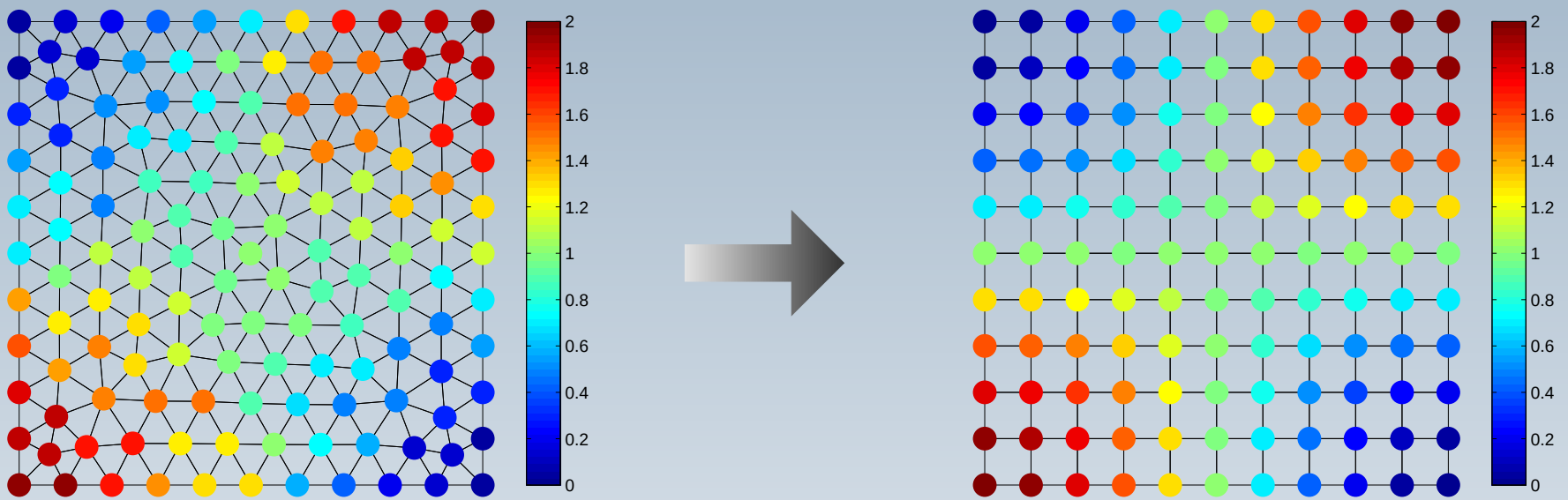


Example: linint2

SAND2011-2450P

linint2: Interpolate from one grid to another grid using bilinear interpolation

$$f(\mathbf{x}) = 1 + \sin(\pi x)\sin(\pi y)$$



Example: linint2

linint2: Interpolate from one grid to another grid using bilinear interpolation

- Read mesh with MOAB and store connectivity
- Define cell topology
- Locate quadrilateral grid nodes ($\mathbf{x}_p$) on triangle mesh
- For each point in triangle grid cell
  - Evaluate basis at point in reference cell

$$\hat{\phi}_i(\hat{\mathbf{x}}_p)$$

  - Transform basis values to physical space

$$\hat{\phi}_i \rightarrow \phi_i$$

  - Evaluate function at point in physical space

$$f^h(\mathbf{x}_p) = \sum_{i=1}^{N} f_i \phi_i(\mathbf{x}_p)$$

Example: linint2

SAND2011-2450P

linint2: Interpolate from one grid to another grid using bilinear interpolation

```
// Get cell topology for base triangle
   ShardsCellTopology cellType(shards::getCellTopologyData<shards::Triangle<3> >() );


// Loop over elements in triangle grid
    for (moab::Range::iterator it = elems.begin(); it != elems.end(); ++it) {

    // Loop over nodes in quadrilateral grid
       for (int ipt=0; ipt<numNodesQuad; ++ipt) {

       // Coordinates of point in physical space
          FieldContainer<double> physPoints(1, spaceDim);
          physPoints(0,0)= quadPoints(ipt, 0);
          physPoints(0,1)= quadPoints(ipt, 1);

       // Define point in reference cell where basis is evaluated
          FieldContainer<double> refPoints (1, spaceDim);
          IntrepidCTools::mapToReferenceFrame(refPoints, physPoints, cellWorkset,
                                              cellType, 0);

       // check whether point is in cell
          double points[2] = {physPoints(0,0), physPoints(0,1)};
          int inCell = IntrepidCTools::checkPointInclusion(points, spaceDim, cellType);
```

Example: linint2

SAND2011-2450P

linint2: Interpolate from one grid to another grid using bilinear interpolation

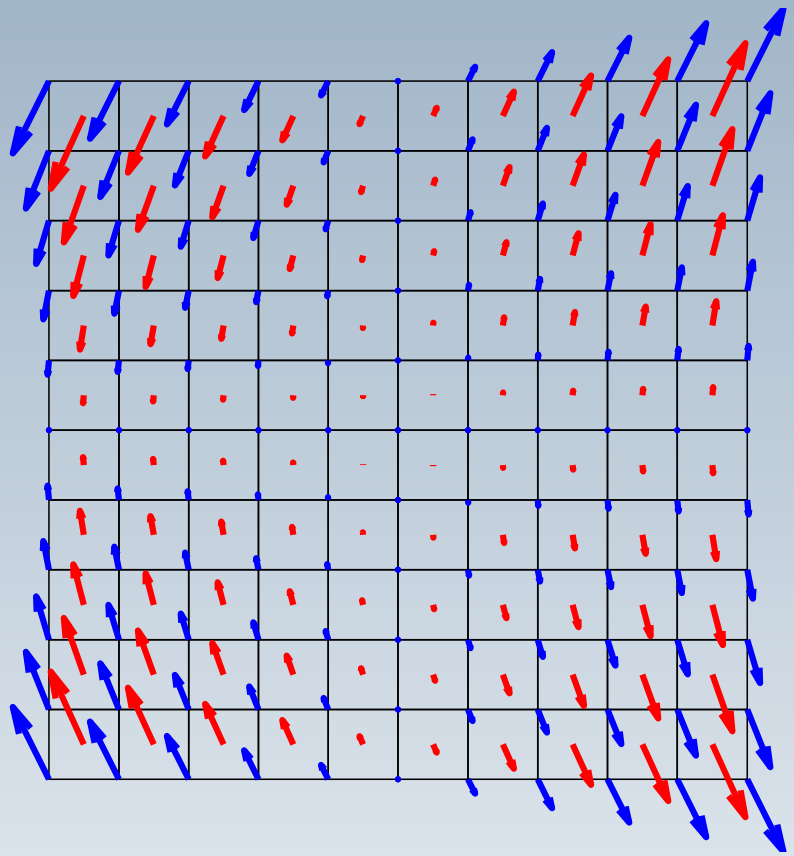
```
if (inCell > 0) {

    // Define basis
      Intrepid::Basis_HGRAD_TRI_C1_FEM<double, FieldContainer<double> > interpBasis;
      int numFields = interpBasis.getCardinality();
      FieldContainer<double> refBasisValues(numFields, 1);

     // Evaluate basis values at points in reference space
        interpBasis.getValues(refBasisValues, refPoints, OPERATOR_VALUE);

     // Containers for basis values transformed to physical space
        FieldContainer<double> physBasisValues (1, numFields, 1);

     // Containers for interpolated values of f
        FieldContainer<double> fNodeQuad (1, 1);

     // Transform basis values to physical frame:
        IntrepidFSTools::HGRADtransformVALUE<double>(physBasisValues,refBasisValues);

     // Evaluate function at a point: \sum f_i(x_p) \phi_i(x_p)
        IntrepidFSTools::evaluate<double>(fNodeQuad, fNodeTri, physBasisValues);

     // Put quad cell values into global array
        fNodeQ[0][ipt] = fNodeQuad(0,0);

} // if point in cell
```

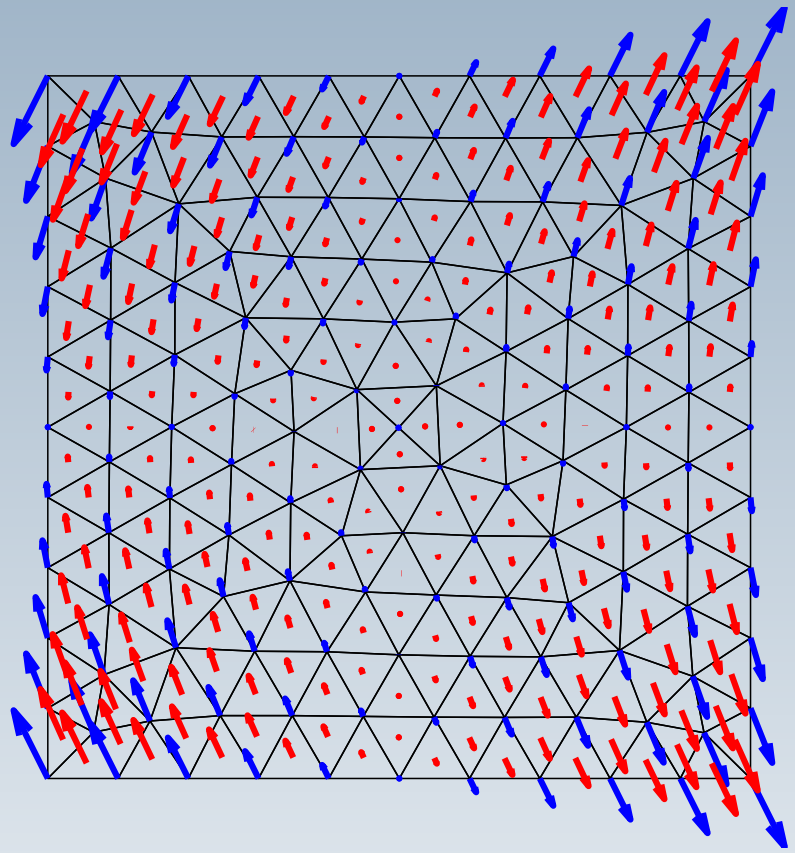
Example: Vector Interpolation

SAND2011-2450P

Interpolate vector from node to cell centers

$$\mathbf{v}(\mathbf{x}) = [xy^2, xy]$$





Example: Vector Interpolation

Interpolate vector from node to cell centers

- Read mesh with MOAB and store connectivity
- Define cell topology
- Define interpolation points on reference cell ($\hat{\mathbf{x}}_p$)
- Evaluate basis at points in reference cell

$$\hat{\phi}_i(\hat{\mathbf{x}}_p)$$

- Transform basis to physical space

$$\hat{\phi}_i \rightarrow \phi_i$$

- Evaluate function at points in physical space

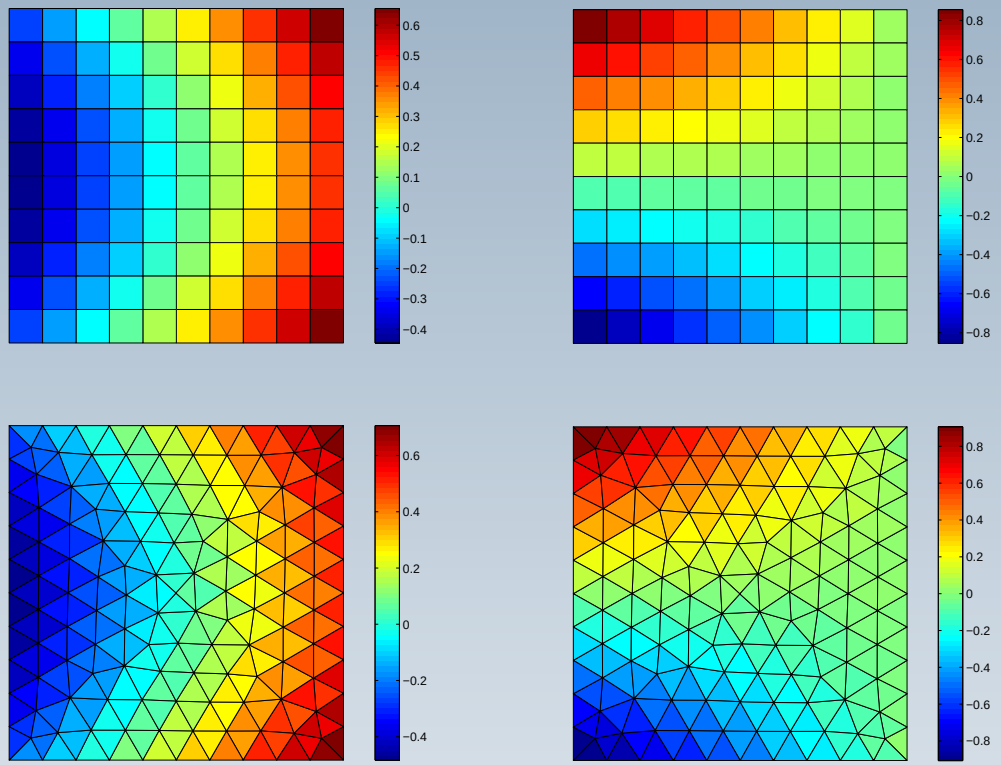
$$\mathbf{v}^h(\mathbf{x}_p) = \sum_{i=1}^{N} \mathbf{v}_i \phi_i(\mathbf{x}_p)$$

Example: Derivatives of a Field

SAND2011-2450P

Calculate divergence and vorticity of nodal vector field at cell centers

$$\mathbf{v}(\mathbf{x}) = [xy^2, xy]$$



divergence

vorticity

Example: Derivatives of a Field

SAND2011-2450P

Calculate divergence and vorticity of nodal vector field at cell centers

- Read mesh with MOAB and store connectivity
- Define cell topology
- Define interpolation points on reference cell ($\hat{\mathbf{x}}_p$)
- Evaluate gradient of basis at points in reference cell ($\nabla\hat{\phi}_i(\hat{\mathbf{x}}_p)$)
- Calculate cell Jacobian (DF)
- Transform gradient of basis to physical space

$$\nabla\phi_i = (DF)^{-T}\nabla\hat{\phi}_i$$

- Evaluate gradient of vector components at points

$$\mathbf{v} = (u, v) \quad \nabla u^h(\mathbf{x}_p) = \sum_{i=1}^{N} u_i \nabla\phi_i(\mathbf{x}_p)$$

- Calculate divergence and vorticity from components

$$\text{div} = \partial u^h/\partial x + \partial v^h/\partial y \quad \text{vort} = \partial v^h/\partial x - \partial u^h/\partial y$$

Example: Derivatives of a Field

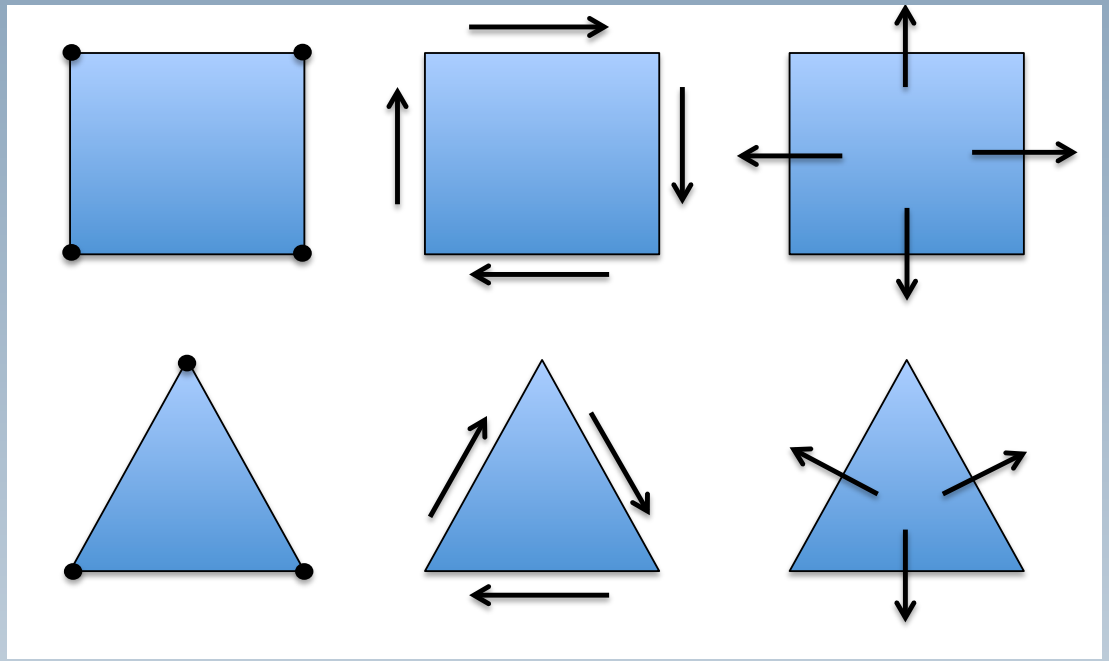
SAND2011-2450P

Calculate divergence and vorticity of nodal vector field at cell centers

```
// Calculate Jacobian
  Intrepid::CellTools::setJacobian(worksetJacobian, evalPoints, cellWorkset, cellType);
  Intrepid::CellTools::setJacobianInv(worksetJacobInv, worksetJacobian );

// Evaluate basis values at evaluation points
  Intrepid::Basis_HGRAD_QUAD_C1_FEM<double, FieldContainer<double> > interpBasis;
  int numFields = interpBasis.getCardinality();
  interpBasis.getValues(basisGrads, evalPoints, OPERATOR_GRAD);

// Transform basis gradients to physical frame
  Intrepid::FunctionSpaceTools::HGRADtransformGRAD<double>(worksetBasisGrads,
                                          worksetJacobInv,   basisGrads);

// Evaluate gradients at a point: \sum v_i(x_p) grad \phi_i(x_p)
  Intrepid::FunctionSpaceTools::evaluate<double>(worksetCell_du, worksetvCoef_x, worksetBasisGrads)
  Intrepid::FunctionSpaceTools::evaluate<double>(worksetCell_dv, worksetvCoef_y, worksetBasisGrads)

// Calculate divergence and vorticity and store in global array
  for(int cell = worksetBegin; cell < worksetEnd; cell++){

   // Compute cell ordinal relative to the current workset
    int cellOrdinal = cell - worksetBegin;

    divCell[0][cell] = worksetCell_du(cellOrdinal,0,0) + worksetCell_dv(cellOrdinal,0,1);
    vortCell[0][cell] = worksetCell_dv(cellOrdinal,0,0 )- worksetCell_du(cellOrdinal,0,1);

  }// *** workset cell loop **
```

Intrepid Flexibility

SAND2011-2450P

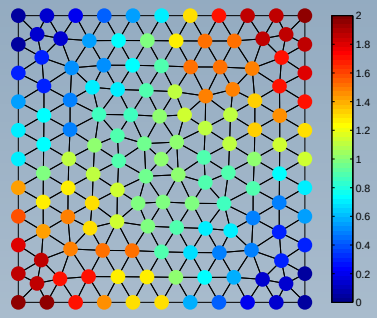


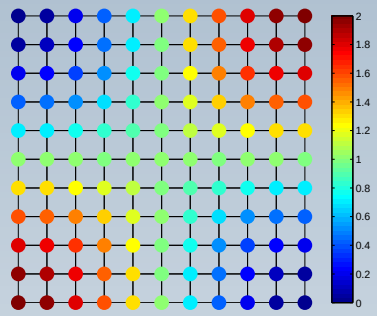
- Previous examples only used nodal basis functions
- Fluxes through a surface can be represented with Raviart-Thomas basis functions
- Other types of data might be better represented with Nedelec (edge) basis functions

Summary and Future Plans

SAND2011-2450P

- Current Capabilities
  - Interpolation from grid to points
  - Function approximations with node, edge, or face basis functions
  - Differential operators
  - Integration over cells
- Future Plans
  - Interpolation from points to grid
  - Additional cell topologies (eg. polygons)





For more information see:
http://trilinos.sandia.gov/packages/intrepid